The DONS rail scheduling system

W.v.d. Aalst
K.M. van Hee
M. Voorhoeve
Eindhoven University of Technology

June 23, 2005

Abstract

This paper describes a system for automated rail scheduling currently being developed at Dutch Rail (NS). This system will be used initially to base infrastructure decisions upon. Rail infrastructure poses constraints upon the rail schedules that can be made within it, because of the large extent in which trains share it. Good infrastructure is the one that allows good schedules. The Petri net based ExSpect editor is used to model infrastructure variants and to help converting them into schedule constraints. With OR techniques, schedules are then made generated. ExSpect is then used to simulate the execution of these schedules within their infrastructure.

Acknowledgements

The authors are indebted to prof. Bertrand (EUT) and dr. Smeets (NS) for initiating and to all DONS team members from NS and BMC for maintaining an interesting cooperation. Most of the ideas within the present paper result from team work; we do not claim credit for them.

1 Introduction

ExSpect [HSV 89,ASPT 94] is a Petri net tool, language and method for the description, analysis and simulation of distributed systems. It has been developed at Eindhoven University and is marketed by Bakkenist (BMC) in Amsterdam. Up to this moment, its practical use has focused around infrastructure decisions of some kind. The infrastructure variants to choose from are described in ExSpect and then simulated to see which one performs best. This brought us into contact with Dutch Rail (NS).

1.1 Infrastructure within NS

As many urban areas in the world, Holland suffers from a transport problem. NS is using its present infrastructure to and even beyond its limits, so investments are needed to remove the present bottlenecks. Because of the long-term character of these investments, various variants have to be compared, by analysis and simulation. However, trains do not arrive and depart randomly, but according to schedules, which are influenced by the future infrastructure. This naturally leads to a system, wherein infrastructure is modeled, schedules then are generated from it, and finally tested by simulation.

We describe briefly the assumptions underlying the schedules made within NS for testing infrastructure. Each scheduled train has an identification and a kind. There are 4 train kinds: intercity (IC), inter-regional (IR) and agglo-regio (AR) plus freight trains (G). Passenger trains depart hourly; if e.g. an IR train with identification '32H' departs at 11:15 from station 'EHV', a (probably different) train with the same identification and kind is supposed to do so at 6:15 and another at 7:15 and so on. The granularity of these schedules is 1 minute.

1.2 Structure of the paper

The present paper describes the DONS system, which we describe by means of the ExSpect terminology. So the latter is briefly introduced to improve understanding. We terminate by discussing the parts realized at present and experiences with it.

2 ExSpect

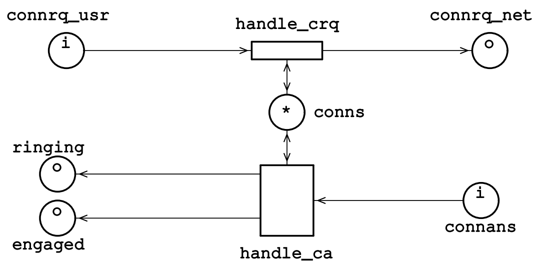
ExSpect is based on Petri nets with colour, time and hierarchy and can be compared to CPN and the Design/CPN tool. Differences is that firing rules are not included in the graphical net using pattern matching, but with Z-like precondition predicates and concurrent assignments. Assignments may be conditional, which makes an atomic subnet (processor) in ExSpect to a group of transitions having complementary preconditions for firing.

The hierarchy can be compared to transition substitution with place fusion. An ExSpect subnet has virtual places or pins that can be fused or installed to real places (or again pins) in the higher net. In installing, the names of the places or pins in the higher net must be matched with the pin names in the subnet, just like one treats function calls in conventional programming languages.

A third feature of ExSpect are stores, which are places that contain a single token throughout their existence.

The following specification exhibits the flavour of ExSpect. We specify part of a telephone network, servicing a set of subscribers. The subnet posesses a store containing the current connections of its subscribers. A connection request is transferred if the requesting subscriber has no current connection. If an answer for the connection request comes from the network, then a connection request must be pending. The answer kind is determined and reacted upon.

The described part is depicted as follows.



The places with the inscriptions are input (i) or output (o) pins, the starred one is a store. The `handle_crq` subnet is a processor without conditional assignments (i.e. a transition). Its description is as follows.

```
proc handle_crq
  [in u : CRQ, out n : CRQ, store s : USR->CON |
   pre not(snd(u) elt dom(s))]
:=
   n <- u, s <- addrq (s, u);
```

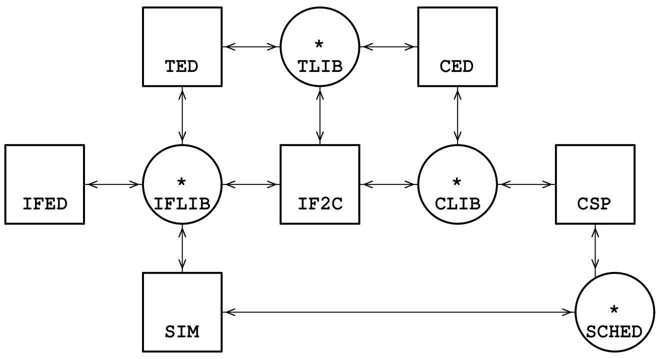
The pin `u` is connected to `connrq_usr` and `n` to `connrq_net`, whereas `s` is connected to `conns`. The functions `not`, `elt` and `dom` are standard functions representing negation, membership and domain respectively. The function `snd` is specific to the telephone example, extracting the sender (of type `USR`) from a connection request (of type `CRQ`). Another specific function `addrq` updates the store. We also give the description of `handle_ca`. This processor has conditional assignments and is in fact a small subnet containing two transitions.

```
proc handle_ca
  [in a : CA, out r : USR, g : USR, store s : USR->CON |
   pre isrequested (s.rcp(a)) ]
:=
   if isengaged(a) then g <- rcp(a) else r <- rcp(a) fi;
```

Of course, `g` is connected to `engaged` and `r` to `ringing`.

3 DONS

The DONS system supports the rail scheduling task. The diverse decisions that planners make are recorded and transformed into constraints for a rail schedule. Then a rail schedule is made satisfying those constraints. If a schedule is impossible, conflicting decisions are reported, allowing the planner to revise them. Finally, a simulation component allows to test generated schedules in more or less realistic circumstances. Below, the architecture of the system is depicted.

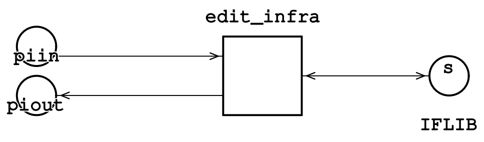


1. system DONS

The system consists of 3 editors: IFED (infrastructure), TED (tabular data) and CED (temporal constraints), whose respective output is stored in IFLIB, TLIB and CLIB respectively. IFED is a Petri-net based editor for rail nets. TED needs infrastructure to match corresponding data against. CED needs the data to match the constraints against. The IF2C transformer generates temporal constraints from infrastructure and tabular data. The CSP program looks at the (temporal) constraints and either detects inconsistencies between them that are reported to the planner or it establishes their consistency by generating a rail schedule satisfying the constraints. The SIM program is Petri-net simulator that simulates the generated schedule being followed in the infrastructure, measuring its stability against disturbance.

3.1 Infra editor

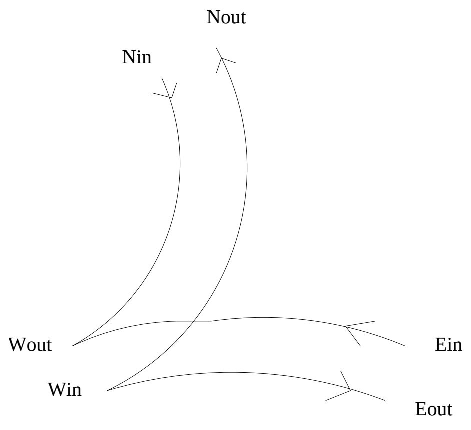
The infrastructure editor is depicted below.



2. Rail infrastructure editor

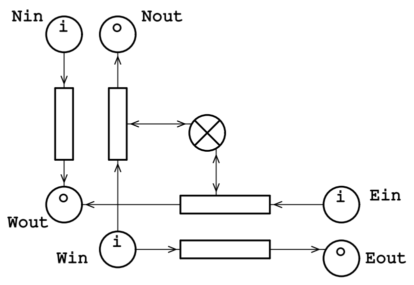
The “piin” and “piout” places are for interaction with the planner. The infrastructure that is specified by the editor has two levels.

On the lowest level nodes of the rail net are specified. Nodes connect various rail sections by means of e.g. switches. The following example depicts a node (called TBA) where three two-track sections meet. The “Win” connection splits into “Nout” and “Eout”, whereas “Nin” and “Ein” are joined into “Win”. The “Ein-Wout” and “Win-Nout” paths have a level crossing, so trains passing these paths have a conflict.



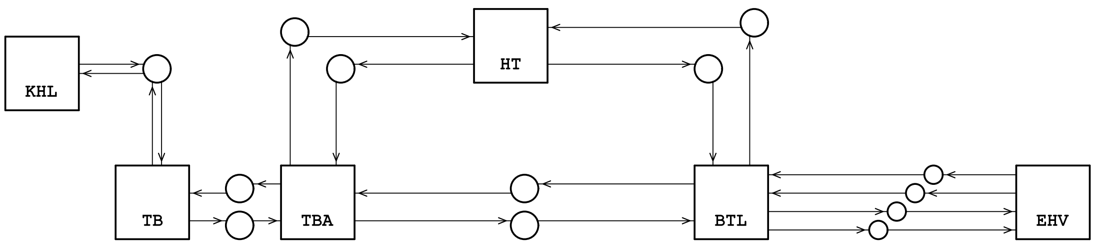
3. Rail node TBA

This situation is translated into a hierarchical Petri subnet by joining connections by “path” transitions if they are connected. These transitions share a conflict store if trains passing the corresponding paths are in conflict.



4. Node TBA as Petri subnet

On the higher levels, nodes are connected by means of rail sections, which can have one or two directions. A rail section is a place connecting two nodes. If the place is input for the one and output for the other, it is an unidirectional section, if it is input/output for both, it is bidirectional. The net below depicts a small part of the rail net in the vicinity of Eindhoven (EHV), as it will be in the future.

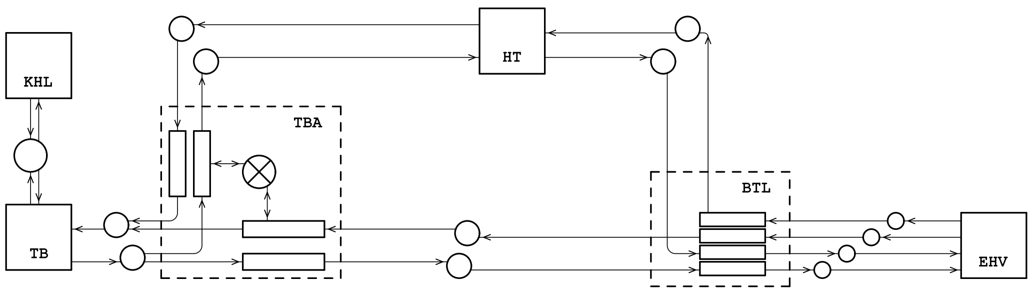


5. Rail net example

From Eindhoven, two sections lead to Boxtel (BTL). There they split; one goes to ’s Hertogenbosch (HT), the other to Tilburg (TB). There is also a section from HT to TB, which joins with BTL-TB near TB in TBA (Tilburg Aansluiting). The TBA node is the one specified above in detail. All sections have their return sections as well.

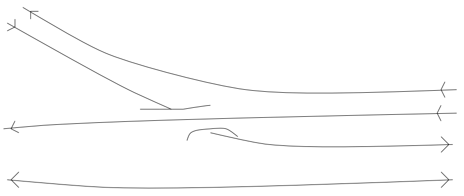
From TB a bidirectional section leads to Kaatsheuvel (KHL), where the famous tourist attraction “de Efteling” is situated. The rest of the Dutch rail net (west of TB, north of HT and south/east of EHV) is omitted. One could add connectors and go up the hierarchy another level.

The nodes TBA and BTL can be unfolded to show the actual connections. Note the absence of conflicts in the node BTL, due to a fly-over planned there.



6. Rail net unfolded

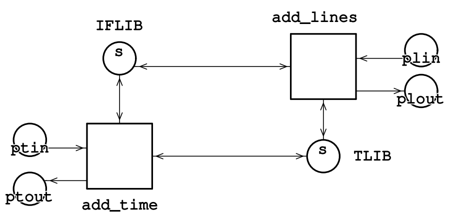
Below is the future situation at the node BTL depicted with the fly-over.



6. Node BTL

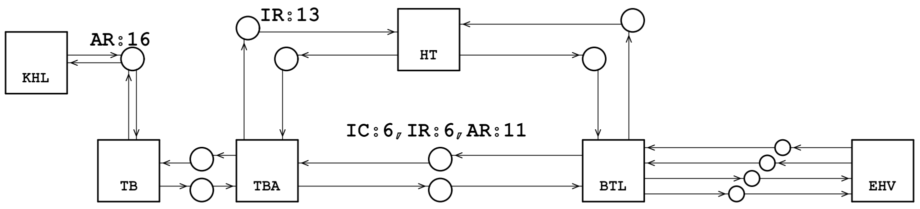
3.2 Tabular data

The tabular data editor extends the infrastructure with tabular information. The same infrastructure component can and will have several tabular data attached to it, so it is modeled as a separate phase, although one and the same editor may be used throughout. There are two different tabular data classes: timing information and line information.



7. Tabular data editor

The timing information gives the minimum times for each train type for a group of rail sections. This information is attached to places in a rail net, as in the following picture.



8. Net extended with timing information

Also the minimum time needed for avoiding conflicts within nodes and rail stretches must be specified. This time depends of the way the control systems are implemented; the default is at present 3 minutes.

The line information gives the required hourly trains with their routes through the net. Two hourly trains constitute a line, one in the forward and one in the backward direction. For example, in the above net, the following lines could be specified.

htid	kind	route
3	IC	EHV,BTL,TBA,TB
5	IC	EHV,BTL,HT
6	IC	EHV,BTL,HT
31	IR	EHV,BTL,TBA,TB
37	IR	HT,TBA,TB
38	IR	HT,TBA,TB
413	AR	EHV,BTL,TBA,TB,KHL
414	AR	TB,KHL
431	AR	EHV,BTL,HT

The table is made by pointing at the successive nodes and is checked against the connection information within the nodes. An hourly train is specified by extending a line with 'H' (Dutch: 'Heen') for the forward and 'T' (Dutch: 'Terug') for the backward direction.

3.3 Transformation

The infrastructure extended with tabular data is converted to temporal constraints. Temporal constraints are built using variables denoting the arrival and departure times of trains within nodes. From the information above, it can be e.g. deduced that 31H will cross TBA 6 minutes after crossing BTL (by default trains run at maximum speed; the possibility of slowing down has to be added explicitly afterward). Also 413H will cross TBA 11 minutes after crossing BTL. These constraints are denoted as follows.

```
31H:TBA - 31H:BTL IS 6
413H:TBA - 413H:BTL IS 11
```

Trains sharing the same rail section must be 3 minutes apart. Hence their departure and arrival times must differ between 3 and 57 minutes modulo 60. Consequently, in our notation,

```
31H:BTL - 413H:BTL IN 3..57
31H:TBA - 413H:TBA IN 3..57.
```

Moreover, they cannot overtake one another, which can be formulated by disallowing some event orders as follows.

```
NOT(ORDER(31H:BTL,413H:BTL,413H:TBA,31H:TBA))
NOT(ORDER(413H:BTL,31H:BTL,31H:TBA,413H:TBA))
```

Here `ORDER(`a, b, c, d`)` is an abbreviation for b `IN` a`..`c `AND` c `IN` b`..`d `AND` d `IN` c`..`a. For bidirectional sections, trains from different directions may not even cross. So

```
ORDER(413H:TB,413H:KHL,413T:KHL,413T:TB)
ORDER(413H:TB,413H:KHL,414T:KHL,414T:TB)
ORDER(414H:TB,414H:KHL,413T:KHL,413T:TB)
ORDER(414H:TB,414H:KHL,414T:KHL,414T:TB)
```

By the conflict in TBA, the crossing of trains in the TB-HT and BTL-TB directions must be 3 minutes apart as well, giving e.g.

```
31H:TBA - 38T:TBA IN 3..57.
```

The consistency checking and transformation of infrastructure and added data into temporal constraints is performed by ASF+SDF ([BHK 89],[HHKR 89]), an integrated language development toolkit. An important feature is that the temporal constraints point back at the planner decisions they were deduced from.

3.4 Constraint editing

The infrastructure constraints deduced above are not the only ones that a schedule must satisfy. In a good schedule, trains for the same direction depart at regular intervals, for example

```
413H:TB - 414H:TB IS 30.
413T:KHL - 414T:KHL IS 30.
```

Also some connections between trains must exist. The train to KHL should maybe connect to the intercity EHV-TB in both directions. The time needed to change trains is judged to be 4 minutes and 7 minutes is the maximum allowed for the connection. This would boil down to the following constraints.

```
414H:TB - 3H:TB IN 4..7
3T:TB - 414T:TB IN 4..7
```

All these so-called marketing requirements are added using a (tabular) constraint editor. A prototype editor is developed using the ASF+SDF environment.

3.5 Constraint manipulation

From the conjunction of all temporal constraints numerous conclusions can be drawn. For instance, it can be deduced from the constraints above that

```
31H:BTL - 413H:BTL IN 3..52.
```

It can even be deduced that the requirements upon the train service TB-KHL are inconsistent: no schedule can meet them all. By applying a few simple rules, most inconsistent constraint sets can be tracked and reported to the planner. He can act to remove inconsistencies; the above inconsistency could be removed by deleting line 414, by removing the regularity requirement or by altering the infrastructure, e.g. by specifiying two unidirectional sections between TB and KHL. Another possibility would be to shorten the time to, say, 12 minutes by using faster trains or omitting intermediary stops.

When all inconsistencies have been removed, schedules can be generated by choosing a value for some variable in accordance with the initial constraints, adding this choice to the constraints and again drawing conclusions from the extended constraint set. The procedure is repeated for every variable that is not yet determined completely by the constraints. If a choice turns out to be inconsistent, backtracking is used.

An alternative for schedule generation is using linear and integer programming techniques. In this case, in addition to meeting the constraints, some goal function can be optimized.

3.6 Simulation

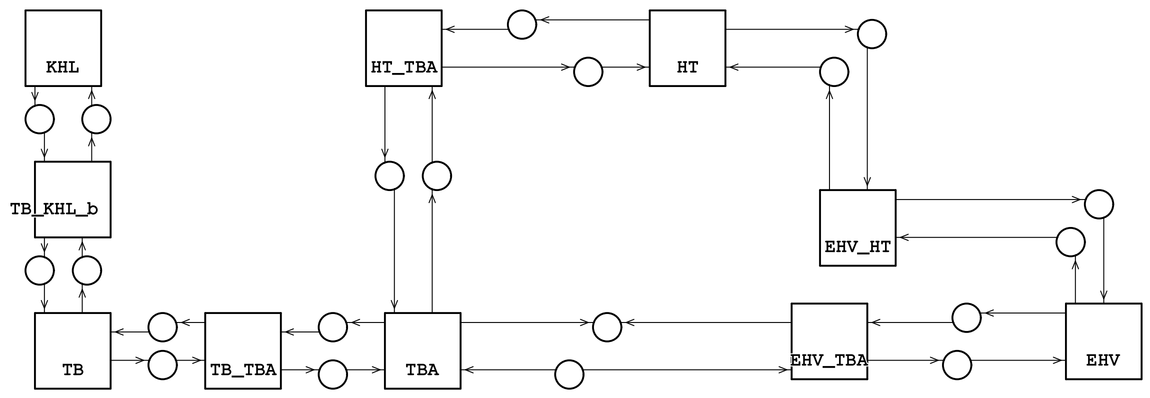
Generated schedules, together with infrastructure can be tested by simulation. One needs to add scenarios, e.g. disturbances of waiting times at stations due to minor technical problems. Decisions have to be made, e.g. priority in case of conflicts at nodes or whether trains wait for their connections or depart according to schedule.

It is possible to specify decision rules in ExSpect and evaluate the performance of infrastructure, schedule and rules together by simulation. ExSpect is even capable of interactive simulation, allowing a human controller to make or overrule decisions.

By testing various schedules in various scenarios, it should be possible to compare different infrastructure variants.

4 Present situation

The system desribed above has not yet been implemented as described above. The present infra editor uses other building blocks, that are somewhat more cumbersome to use. The example net above would look as follows.



9. Net specified in current system

The main difference is that rail stretches are components themselves instead of a pair of places. This on average doubles the number of objects to input. Adding the tabular data is done with ASCII editors; their output is transformed by the ‘awk’ program into ExSpect format. The temporal constraints are then generated by running the ExSpect system: the information about trains and their timing is given to and extracted from the components by tokens representing trains and finally stored into a file.

The marketing requirements are then added using a text editor and stored in a database. The constraint syntax is limited to expressions of the form

v_1 `-` v_2 `IN` c_1`..`c_2,

where the v-s are variables and c-s constants. These constraints are then manipulated very fast, testing them for consistency within seconds and generating schedules within a few minutes. This is done by the CADANS program developed by Schrijver and Steenbreek (CWI Amsterdam). Simulation with scenarios and decision rules has not been implemented yet.

4.1 Problems encountered

Initially, NS was surprised and delighted that automated schedule generation was indeed possible. In the old situation, making an infrastructure proposal and a schedule was done by hand, solving problems as they appeared, making it nearly impossible to make several alternative schedules for the same infrastructure variant. When the initial enthousiasm was over, several shortcomings of the system were recognized however.

While specifying infrastructure, a planner's decision corresponds to several manipulations with the editor. A special-purpose rail net editor would have done a better job than the general-purpose Petri net editor extended with a library of building blocks.

In adding tabular data, consistency checks with the target infrastructure are limited. The generation of temporal constraints is not very fast for large nets, moreover minor modifications to the infrastructure or data require a regeneration of all constraints, instead of a local update.

The link of temporal constraints with the planners decisions that led to them has been lost in the generation process, so the temporal constraints themselves have to be interpreted.

Finally, the platform used is a SUN workstation with Open-Windows. The usergroup is instead used to PC's with MS-Windows. This problem will be solved in the near future when ExSpect becomes available on PC platforms.

4.2 Contribution of Petri net theory

The role of Petri net theory in DONS has been very small so far. What has been proved is that rail nets with their static structure can be adequately modeled as Petri nets. The possibility for modeling and simulation within a graphical context has been one of the reasons for NS to carry out the DONS project with ExSpect.

Generally, Petri nets with colour and hierarchy allow for a DFD-like modeling method that is very attractive to non-experts, while still retaining a formal semantics. The existence of an ASCII-based description makes it possible to transform models by tools like ASF+SDF for further processing by other software.

There will be opportunities for more advanced theory in the future. In modeling rail nets with their control structures, some interesting questions arise that could be answered by Petri net theory. These questions adress safety (e.g. prove that trains cannot collide) as well as performance (e.g. the capacity of a controlled net).

Seen in this light, the DONS project is another step towards applying Petri net theory to traffic problems in practice.

5 References

ASPT 94
ExSpect User Manual (Rel 4.2) Bakkenist Management Consultants 1994

HSV 89
K.M. van Hee, L. Somers and M. Voorhoeve
Executable specifications for distributed information systems
Proceedings IFIP WG 8.1 Working Conference pp.139-156 North-Holland 1989

BHK 89
J.A. Bergstra, J. Heering, P. Klint (eds)
Algebraic Specification
ACM Frontier Series, Addison-Wesley 1989

HKHR 89
J. Heering, P.R.H. Hendriks, P. Klint and J. Rekers
The syntax definition formalism SDF - Reference Manual
SIGPLAN Notices 24(11): pp.43-75, 1989